AUTOMATICALLY CREATING JAVASCRIPT OBJECTS TO INVOKE METHODS ON SERVER-SIDE JAVA BEANS

Inventor

Robert DeSantis

Background

Technical Field

[0001] The present invention generally relates to the JavaScript programming language operating in a J2EE server environment, and more specifically to a method and system for automatically creating and delivering JavaScript objects to invoke methods on server-side Java beans without the need for additional installations on the client browser machine.

Background of the Invention

[0002] A common problem is distributed applications, and specifically in Web applications, is how to invoke server-side code from a remote client (Web browser). Conventional solutions such as CORBA (Common Object Request Broker Architecture), RMI (Remote Method Invocation), and SOAP (Simple Object Access Protocol) all address this problem. Each of these has disadvantages however, a primary disadvantage being their complexity. Other disadvantages include the requirement for an IDL (interface definition language), and additional components that need to be installed into the browser environment. For example, a client-side program has to know which parameters are required to make a call to a standard server-side servlet (such as a JSP (Java Server Page)). These parameters must be explicitly coded as a URL parameter, so that every time a server-side servlet and/or bean is changed, the client-side call must be

re-programmed. In other words, the basic code must be written both on the server and the client to wire client-side calls to server-side beans. Moreover, not all remote clients should be allowed to invoke all server-side methods.

[0003] What is needed is a simple way to invoke a method on server-side Java beans that also maintains security features relating to who can invoke the methods without additional installations and other complexity on the client.

Summary of the Invention

[0004] The described embodiments of the present invention simplify the use of server-side methods. The described embodiments require no client installation when using Internet Explorer 5.5 and above. Other embodiments, of course, support other browsers. No developer or user setup is required on the client and no IDL needs to be written or maintained.

[0005] A preferred embodiment of the invention examines a server-side Java bean and automatically generates in real-time a JavaScript representation to provide programmatic access to the server bean by a client. The JavaScript representations of the methods are sent to the Web browser. The browser then creates JavaScript objects that can be called directly from other JavaScript code on the client side.

[0006] A developer of JSP (Java Server Pages) identifies which session-scoped server beans he wants to be available on the client. The developer can also identify exactly which methods from those beans should be made available. Once the beans have been identified, the server (or other appropriate data processing system) generates JavaScript source code for each bean, which is then added to the page delivered to the

client's browser. The generated source creates an object on the client with the same name as the server-side bean and provides proxy methods for all of the previously identified server-side bean methods. The net result is that the JavaScript code can simply invoke server-side bean methods.

Brief Description of the Drawings

[0007] Fig. 1 is a block diagram showing a client and a server data processing system.

[0008] Fig. 2 is a flowchart showing a server-side process.

[0009] Fig. 3 is a flowchart showing a client-side process.

[0010] Fig. 4a is a flowchart showing automatic generation of JavaScript by a server-side.

[0011] Fig. 4b is a functional description of an automatically generated JavaScript file.

[0012] Fig. 5 is a block diagram showing two clients and a server data processing system where the two clients are allowed to access different server-side methods.

[0013] Fig. 6a provides an example of a full JSP page.

[0014] Fig. 6b provides an example of a browser display generated by the JSP page of Fig. 6a.

[0015] Fig. 7 provides an example of a server-side bean.

[0016] Fig. 8 provides an example of an automatically created JavaScript.

[0017] Figs. 9a and 9b show an example of an HTML page sent to the client side, the HTML page including the automatically created JavaScript.

Detailed Description of Preferred Embodiments

A. Background:

[0018] JavaScript is an interpreted script language from Netscape Communications Corporation, based in Mountain View, California. JavaScript code can be imbedded in HTML pages and interpreted by the Web browser or client.

[0019] Java Server Page (JSP) is a server-side technology for generating servlets, small programs that produce a Web page and run on the server.

[0020] A browser enabled for DHTML (Dynamic HTML) treats each page element (division or section, heading, paragraph, image, list, and so forth) as an "object." For example, each heading on a page can be named, given attributes of text style and color, and addressed by name in a small program or "script" included on the page. This heading or any other element on the page can be changed as the result of a specified event such a mouse passing over or being clicked or a time elapsing.

B. Embodiments:

[0021] Fig. 1 is a block diagram showing a client and a server data processing system. The system includes a client 100 (such as a Web browser) and a server 110 (which can be any standards-based Java application server). Client 100 and server 110 are preferably connected via a network, such as the Internet or an intranet, although they can be connected using any appropriate connection that allows the transfer of data. In the example, server 110 includes several server-side beans. Particularly, server 110 includes several Java beans (beans1, beans2, beans3) including corresponding methods. For example, bean1 114 includes a method M1. (While bean2 and bean 3 may also have methods, these are not shown for simplicity of example). A developer written JSP,

available from server 110, includes a clientBean JSP tag describing which bean(s) and methods to be made available on the client. As will be discussed below, server 110 automatically creates a JavaScript file corresponding to bean1 and its method M1 and sends the JavaScript file along with the HTML page to the client 100 so that the client can easily invoke server-side methods. In the described embodiment, the JavaScript file is a separate file, and a reference to the generated JavaScript file is included in the HTML page. This is all a result of including clientBean JSP tag 112 in the JSP page.

[0022] Fig. 1 shows transfer 120 of the JavaScript file to client 100. This JavaScript will be executed on client 100 to create a client-side object 103 corresponding to server-side bean1 114. This client-side Java object 103 can then be invoked by client-side software, causing a call 130 from client 100 to server-side method M1. The server-side methods may return results 140.

[0023] The described embodiments of the present invention simplify the use of server-side methods. A preferred embodiment of the invention examines a server-side Java bean that is registered with the server (such as bean1) and automatically generates a JavaScript representation of the methods in the bean. The JavaScript representations of the methods are sent to the Web browser 100. The browser then creates JavaScript objects 103 that can be called directly from other code on the client side.

[0024] On the server, a developer of JSP (Java Server Pages) identifies which session-scoped server beans (such as bean1) he wants to be available on the client. The developer can also identify exactly whether all or some methods from those beans should be made available. Once the beans have been identified, the server generates JavaScript source code for each bean, which is then added to the page delivered to the client's

browser. The generated source creates an object on the client with the same name as the server-side bean and provides proxy methods for all of the previously identified server-side bean methods. The net result is that the JavaScript code can simply invoke server-side bean methods.

[0025] Fig. 2 is a flowchart showing a server-side process. In the described embodiment, a JavaScript file is created for each unique combination of bean and bean methods to be delivered to each client. Thus, each of a plurality of clients will receive an individualized JavaScript that controls which server-side method that client will be able to access on each exposed bean. More details are shown in Fig. 5.

[0026] Initially, the Java beans are instantiated on the server side using conventional techniques, such as those techniques familiar to users of Java 2, Enterprise Edition (J2EE). For each session, beans are registered 200. An API for the server-side methods results from the instantiation 210. It will be understood that these beans can be obtained from various sources, such as customers, or from the owner of the server 110.

[0027] For each session, beans are registered 200. In the described embodiment, this entails:

[0028] 1) creating a JSP tag header 112 by a human programmer as shown in Fig. 1,

[0029] 2) adding the term "clientBean" to the header specifying the name of the bean,

[0030] 3) specifying the specific methods to be available. This third step is optional.

[0031] Registration of beans creates corresponding data structures storing the results of the registration.

[0032] As an example, a blox header that does not specify which methods of a bean can be accessed might look like this:

```
<blox:header>
        <blox:client Bean name = "bean1">
</blox: header>
```

In this example, all methods of bean1 are exposed to the client.

[0033] As another example, a blox header that specifies which methods of a bean can be accessed might look like this:

```
<blox:header>
        <blox: clientBean name = "bean1">
                <blox: method name = "M1">
                <blox: method name = "M2" protect = "T"/>
</blox: header>
```

[0034] In this example, method M1 of bean1 is exposed to being called from the client. Furthermore, in this example, a "protect" parameter for a method M2 is explicitly set to TRUE. Thus, in this example, the method M2 of bean1 cannot be accessed.

[0035] It should be noted that the registration process is accomplished using JSP "tags" that are familiar to programmers in the JSP and JavaScript environments. Thus, being able to register a bean and its methods using <blox:header>, <blox:clientBean>, and <blox:method> tags feels very familiar and easy to understand to the JSP programmer. It should be understood that other syntaxes could be used to indicate that beans and their methods should be registered as available to the client.

[0036] In element 220 of Fig. 2, as a result of the registration, server 110 automatically generates a JavaScript file corresponding to the registered methods (i.e., the methods exposed/allowed to the client). The name of the JavaScript file is uniquely

identified based on the bean name and the allowed methods. The JavaScript file preferably has a cyclic redundancy check (CRC) code computed from the method list if and only if methods were specified at registration to uniquely identify the generated JavaScript file. If no method were explicitly specified, then there is no CRC added to the JavaScript name. In a preferred embodiment, if the JavaScript file was previously created, then server 110 uses the previously created file if it was cached on the server (or other appropriate place). Details of element 220 creating a JavaScript file are shown in Fig. 4.

[0037] In element 230, server 110 adds the created JavaScript file to a Web page (as, for example, a JavaScript include) and sends the web page to a client upon request using a conventional mechanism. In at least one embodiment, the JavaScript is added using a <script> tag. In another embodiment, the JavaScript is added using an include of the JavaScript file.

[0038] Fig. 3 is a flowchart showing a client-side process. In element 300, the browser loads a Web page containing the automatically generated JavaScript. In element 310, the JavaScript is automatically executed using conventional mechanisms to instantiate a JavaScript object of the same name as the server side bean. The JavaScript object includes all of the specified methods of the server-side bean (which may be all). In element 320, the JavaScript object is added to the browser's document object using conventional mechanisms. Elements 310-320 occur due to the standard ways that a browser handles a JavaScript file.

[0039] The result of the process of Fig. 3 is that user-written JavaScript code on the client side can invoke methods on the server-side bean through the instantiated JavaScript

object. It is important to note that the present invention requires no changes, installations, or modifications to the client side or browser. Thus, there is zero installation when using browsers equipped to work with the invention (such as, for example, Internet Explorer 5.5 and above). No developer or user setup is required on the client.

[0040] When the bean method is called from JavaScript on the client side, Java objects returned from the bean methods called on the server are converted to JavaScript objects on the client. This works in both directions. JavaScript objects passed to the method are converted to Java objects when the method is called. In addition, Java exception(s) are also returned from the server to the client.

[0041] Fig. 4a is a flowchart showing automatic server-side generation of JavaScript. Fig. 4b shows the functions that are included within the JavaScript 102. Element 400 performs introspection on the Java bean to which methods exist (and have been registered as available to the client). Element 410 inserts JavaScript code to set up a SOAP request for each server-side bean method. Element 420 inserts JavaScript code to instantiate a client-side JavaScript object into the HTML page generated by the JSP. An example of a JavaScript is shown in Fig. 8, discussed below.

[0042] Fig. 4b shows the functions that are included within the JavaScript 102. As shown in the example of Fig. 8, these include a declaration of public methods for this session, Soap requests for each server-side method, and instantiation of a client-side JavaScript Object corresponding to the server-side object. Fig. 5 is a block diagram showing two clients and a server data processing system where the two clients are allowed to access different server-side methods of a bean (i.e., where different methods of the bean are registered for different client sessions). In the example, a server-side bean

bean1 has two methods (M1 and M2). In this example, two JavaScript files are automatically created to allow two different clients to access different subsets of the methods from the same bean class. Specifically, client 100 can access method M1 and client 101 can access method M2. In the described embodiment, the name of Javascript#1 includes a CRC string based on the name M1 and the name of JavaScript #2 includes a CRC string based on the nameM2. The use of a CRC serves to differentiate the JavaScript names and aids in the security of the system.

[0043] Fig. 6a provides an example of a full server-side JSP page 600 that demonstrates how to create and register a server-side beanThis is one of many implementations of ways in which a bean can be registered with a J2EE server. Section 602 demonstrates an example of creating and registering a J2EE session bean. Section 604 demonstrates an example of registering the bean as a client bean so that it can be called from the client 100. Section 606 invokes the method of the bean by calling the server-side bean's method and displaying the return value from the server-side bean method.

[0044] Fig. 6b provides an example of a browser display generated by the JSP page of Fig. 6a. The buttons visible are those defined in the JSP page and cause the browser to invoke JavaScript which in turn calls the server-side bean methods.

[0045] Fig. 7 provides an example of source for a server-side bean named MyBean having two methods setBlox and showDataLayout. Note that this bean is the bean registered in element 604 of Fig. 6.

[0046] Fig. 8 provides an example of an automatically created JavaScript file 102 for the bean MyBean of Fig. 7. Corresponding to the functionality shown in Fig. 4b, an

element 802 declares methods of a server-side bean as public. In the described implementation, elements 804 and 806 set up SOAP requests for each server-side method for this session. The SOAP standard is well-known and is described in "SOAP Version 1.2 Part 1: Messaging Framework" Dec 19, 2002, which is available from W3C at http://www.w3.org/TR/2002/CR-soap12-part1-20021219/, and which is herein inorporated by reference. It will be understood that any appropriate protocol, including both public and proprietary protocols, can be used in place of SOAP to pass parameters and results without departing from the spirit of the present invention.

[0047] Element 808 instantiates a client object and will be executed on the client. Element 810 adds the object to the browser's document object and will be executed on the client.

[0048] Figs. 9a and 9b show an example of an HTML page sent to the client side, the HTML page including the automatically created JavaScript. In this embodiment, the HTML page is a rendered JSP page. The HTML page has a series of "included" files. A first included file 902 is a JavaScript file named Soap.js. An example of this file is shown in Table 1. This file implements the SOAP protocol to calls to and returns from the server-side methods. A second included file 904 is a JavaScript file named BloxAPI.js. An example of this file is shown in Table 2. A third included file 906 is a JavaScript file named bean_myBean.js. This is the automatically generated JavaScript file shown in Fig. 8. Tables 1 and 2 are a part of this specification and are herein incorporated by reference.

[0049] In Fig. 9b, which is a continuation of the HTML page, element 912 calls the server-side bean's method and displays the return value from the server-side bean method.

Element 914 invokes the client method showDataLayout using an input of true and an input of false. In accordance with the invention, these parameters are passed to a server-side method called showDataLayout (see method in server-side bean of Fig. 7).

[0050] In the described embodiment, the automatically generated JavaScript file performs type conversion so that data sent to the server is in an expected format and data received from the server is converted to an expected format. As will be understood by those familiar with the art, the invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims and equivalents.